US006351816B1

(12) **United States Patent**
Mueller et al.

(10) **Patent No.: US 6,351,816 B1**
(45) **Date of Patent: *Feb. 26, 2002**

(54) **SYSTEM AND METHOD FOR SECURING A PROGRAM'S EXECUTION IN A NETWORK ENVIRONMENT**

(75) Inventors: **Marianne Mueller**, Woodside; **David Connelly**, Los Altos, both of CA (US)

(73) Assignee: **Sun Microsystems, Inc.**, Palo Alto, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **08/949,042**

(22) Filed: **Oct. 10, 1997**

Related U.S. Application Data

(62) Division of application No. 08/652,703, filed on May 19, 1996.

(51) **Int. Cl.**[7] **H04L 9/00**
(52) **U.S. Cl.** **713/201**
(58) **Field of Search** 713/200, 201, 713/202; 380/3, 4, 23, 25, 28, 29; 709/203, 223, 305

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,163,091	A *	11/1992	Graziano et al.	380/25
5,191,613	A *	3/1993	Graziano et al.	380/25
5,289,540	A *	2/1994	Jones	380/4
5,341,477	A *	8/1994	Pitkin et al.	395/200
5,381,332	A *	1/1995	Wood	705/8
5,452,454	A *	9/1995	Basu	709/203
5,475,819	A *	12/1995	Miller et al.	709/223
5,475,826	A *	12/1995	Fischer	714/6
5,689,708	A *	11/1997	Regnier et al.	395/682
5,742,759	A *	4/1998	Nessett et al.	713/201
5,742,762	A *	4/1998	Scholl et al.	707/203
5,761,663	A *	6/1998	Lagarde et al.	707/10
5,926,631	A *	7/1999	McGarvey	707/204
5,928,323	A *	7/1999	Gosling et al.	709/203
5,933,503	A *	8/1999	Schell et al.	380/25
5,956,733	A *	9/1999	Nakano et al.	707/204
5,958,051	A *	9/1999	Renaud et al.	713/200
5,983,348	A *	11/1999	Ji	713/200
6,009,176	A *	12/1999	Gennaro et al.	380/25

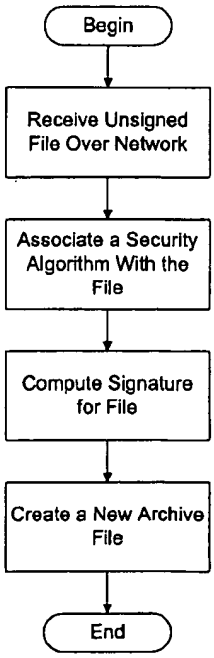
* cited by examiner

Primary Examiner—Dieu-Minh Le
(74) *Attorney, Agent, or Firm*—Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P.

(57) **ABSTRACT**

A system and method is provided for managing security on a server that receives code for execution. A security manager resides on a server and determines whether to permit the execution of a servlet based on a characteristic of the servlet. The security manager makes this determination by performing a number of security checks implemented as a security policy that is configured based on the servlet's network source.

10 Claims, 4 Drawing Sheets



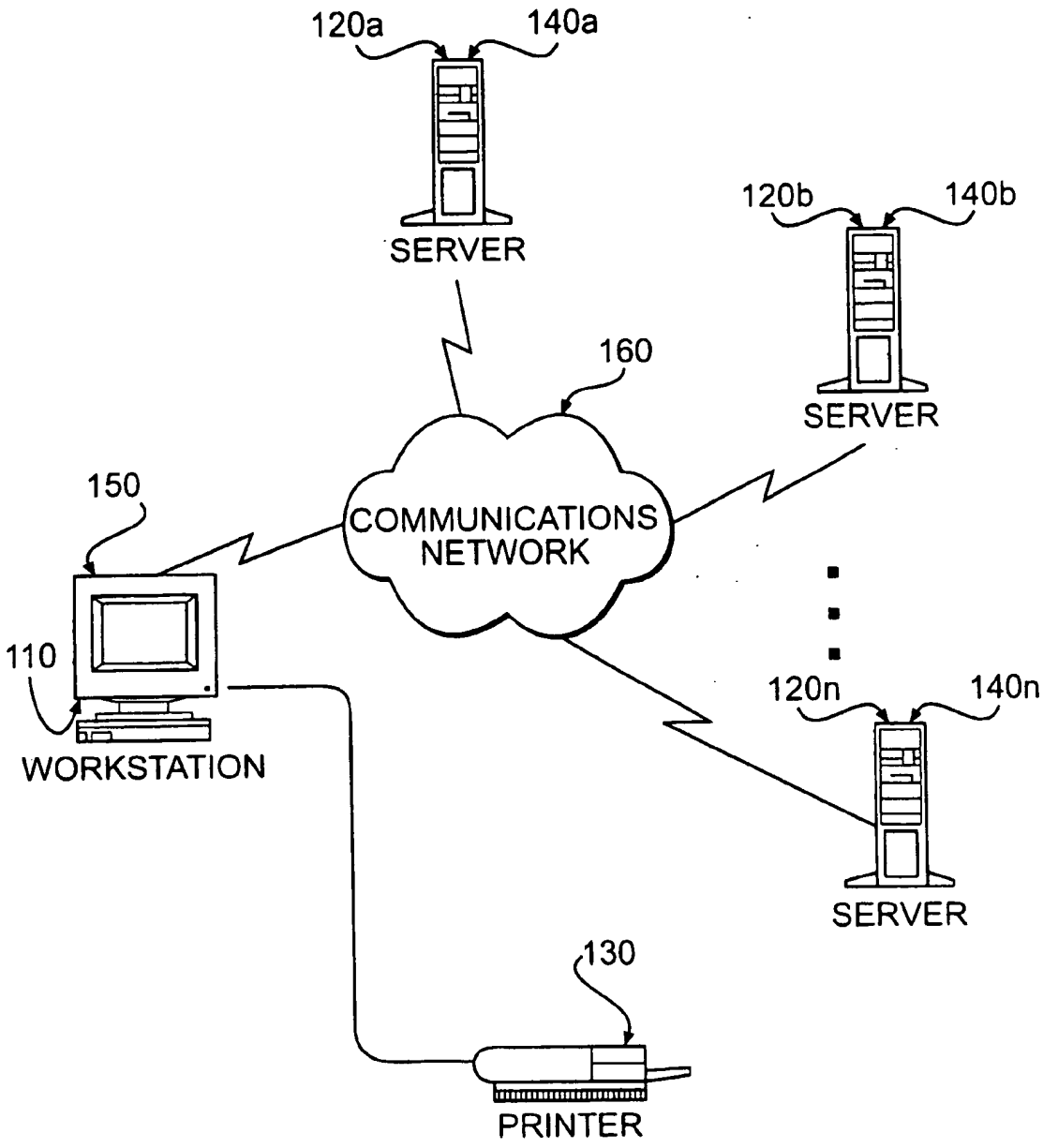


FIG. 1
PRIOR ART

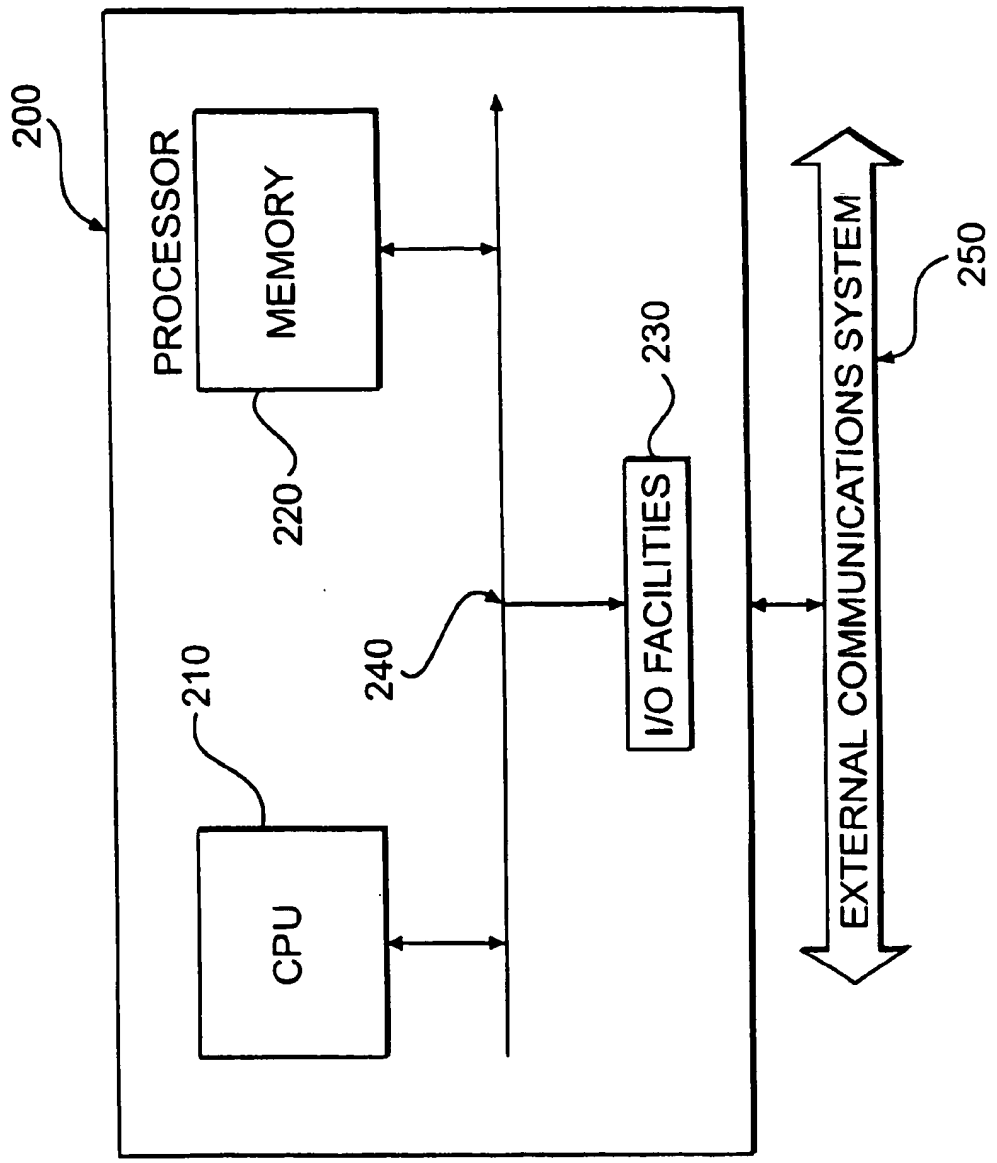

200
PROCESSOR
MEMORY
220
230
I/O FACILITIES
240
210
CPU
EXTERNAL COMMUNICATIONS SYSTEM
250

FIG. 2

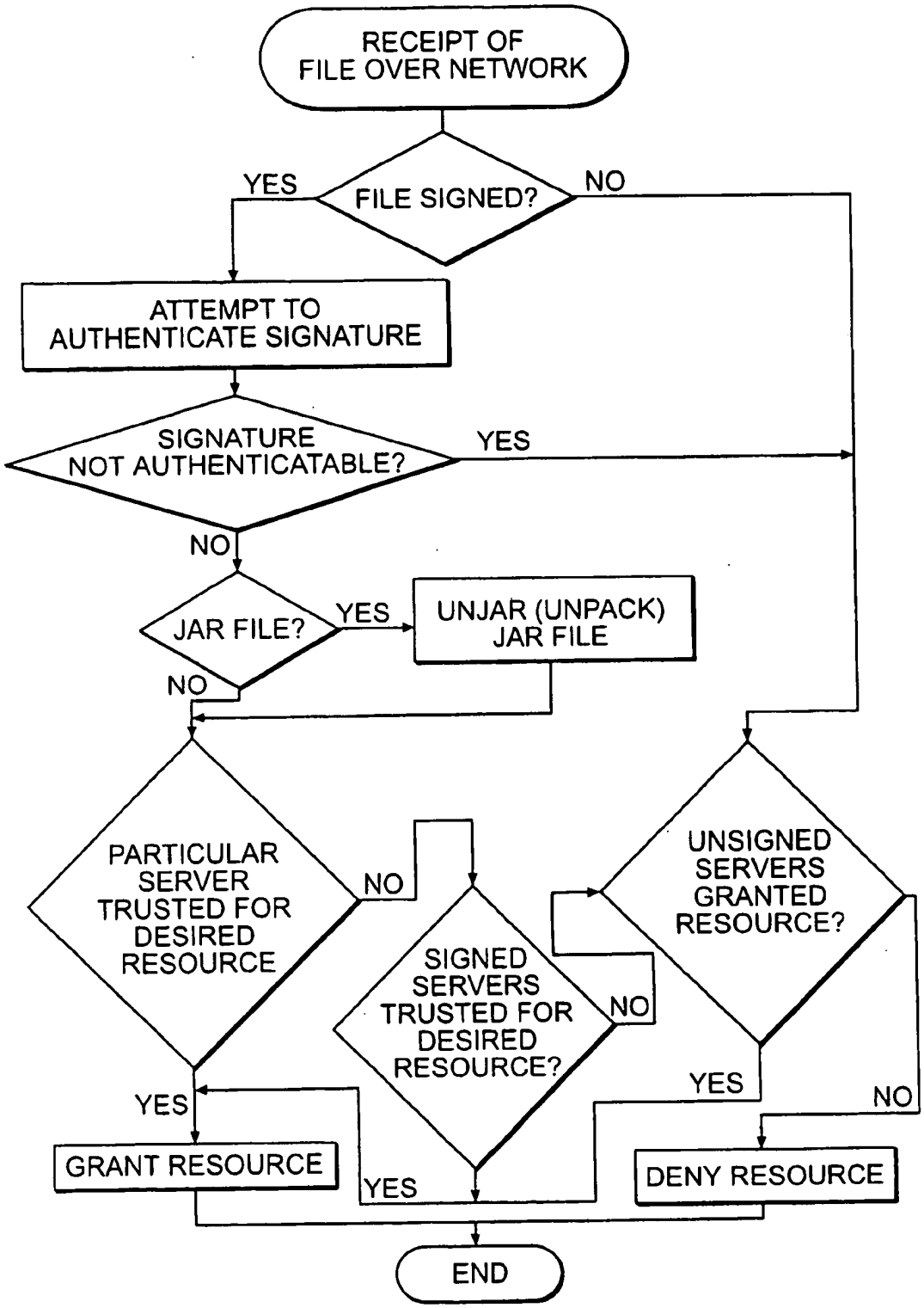

RECEIPT OF FILE OVER NETWORK
FILE SIGNED?
YES
NO
ATTEMPT TO AUTHENTICATE SIGNATURE
SIGNATURE NOT AUTHENTICATABLE?
YES
NO
JAR FILE?
YES
UNJAR (UNPACK) JAR FILE
NO
PARTICULAR SERVER TRUSTED FOR DESIRED RESOURCE
NO
SIGNED SERVERS TRUSTED FOR DESIRED RESOURCE?
NO
UNSIGNED SERVERS GRANTED RESOURCE?
YES
NO
YES
GRANT RESOURCE
YES
DENY RESOURCE
END

FIG. 3

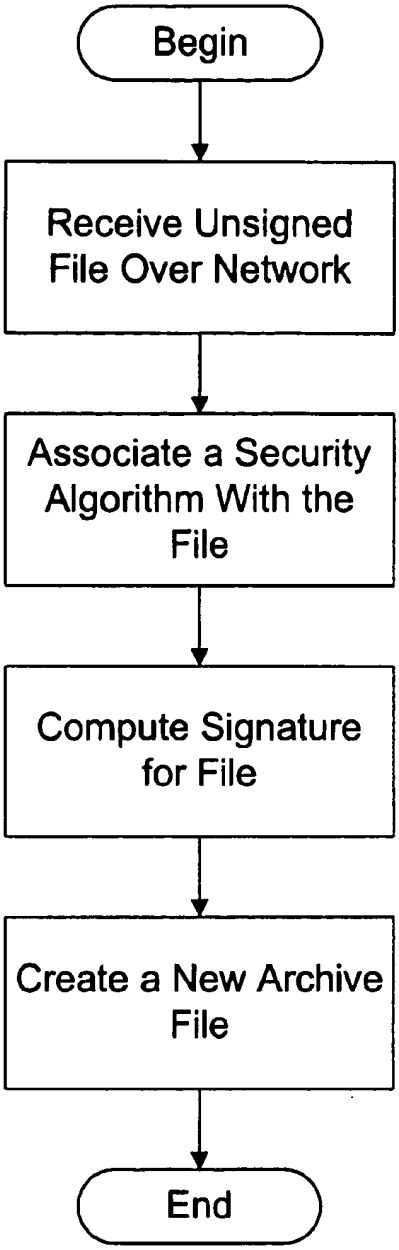

Begin
Receive Unsigned File Over Network
Associate a Security Algorithm With the File
Compute Signature for File
Create a New Archive File
End

FIG. 4

SYSTEM AND METHOD FOR SECURING A PROGRAM'S EXECUTION IN A NETWORK ENVIRONMENT

This is a request under 37 CFR 1.60 for filing a Division of application No. 08/652,703, filed May 30, 1996.

BACKGROUND OF THE INVENTION

This application relates to the provision of services in a client-server context. More particularly, this application relates to securing inter-server services on behalf of a client over a network.

FIG. **1** illustrates a typical client-server environment within the World Wide Web. As one of ordinary skill in the art will readily appreciate, a user's accessing a web page on the World Wide Web involves the cooperation of (at least) two pieces of software: the web browser **110**, typically directly under the user's control as software on the workstation **150**, and the server **120** for the web page. Responding in a manner predetermined by the author of the web page to transactions initiated by the browser **110**, the server **120** typically resides on a separate processor **140**.

FIG. **2** sketches a processor **200** such as a workstation **150** or server **120**. Such a processor includes a CPU **210** to which a memory **220** and I/O facilities **230** connect by a bus **240**. The processor **200** connects to an external communications system **250** which is, for example, a network or modem communications link and memory **220** includes programs **260**. Programs **260** may include one or more programs. Although programs **260** are depicted as being stored in memory **220**, one skilled in the art will appreciate that all or part of programs **260** may be stored on or read from other computer readable media, such as secondary storage devices **270**, like hard disks, floppy disks and CD-Rom, a digital signal received from a network such as the Internet, or other forms of RAM or ROM, either currently known or later developed.

As the HyperText Markup Language (HTML) is the preferred language for authoring web pages, the description below is in the terms of HTML. These terms are explained in, for example, I. S. Graham, *The HTML Sourcebook*, 1996 (John Wiley & Sons, Inc., 2d Edition). Graham is incorporated herein by reference to the extent necessary to explain these terms. However, Graham is not prior art.

In addition to text and static images for display on the user's workstation **150** via the user's browser **110**, a web page can also include an applet. An applet is a program included in an HTML page, whose execution a user can observe via a browser **110** enabled to recognize, download and execute the applet and to display the results of the applet's execution. The HotJava® browser, available from the assignee of the instant invention, is the preferred browser **110**, and the Java® environment, also available from the assignee of the instant invention, is the preferred environment for encoding and executing applets.

The Java® environment is described in, for example, *Java® Unleashed* (Sams.net Publishing, 1996). *Java® Unleashed* is incorporated herein by reference to the extent necessary to explain the Java® environment. However, *Java® Unleashed* is not prior art.

An applet typically is a small program residing on a server **120**. Some HTML document refers to the applet using the <applet> tag. When a browser downloads the HTML document and recognizes the <applet> tag, it also downloads the applet identified by the applet tag and executes that applet.

Written in a general purpose language such as Java®, an applet is in this way unrestrained in its functionality. It can perform any function which a program written in any other general purpose language (such as C or PL1) can accomplish. The methodologies of applets, however, are constrained by the Java® environment in order to minimize the security risks an applet presents to the workstation **150**. That is to say, an applet is restricted to "play" within a bounded "sandbox."

While a security policy may suffice for the transfer of code from a server to a client, the transfer of code for execution from one server to another server presents greater security risks and requires a more stringent security policy. Accordingly, there is a need for a managing security on a server which receives code for execution.

SUMMARY OF THE INVENTION

Herein is disclosed, in a network environment, a security manager residing on a server and deciding whether to permit the execution of a servlet based on a characteristic of the servlet.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a typical client-server environment within the World Wide Web.

FIG. **2** sketches a processor such as a workstation or server.

FIGS. **3, 4** are flowchart illustrating the flow of command according to an embodiment of the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

U.S. patent application Ser. No. 08/657,712 (filed May 30, 1996, entitled, "Method and System for Facilitating Servlets," naming as inventors Pavini P. Diwanji et al., and assigned as well to the assignee of the instant patent application) describes a servlet. U.S. patent application Ser. No. 08/657,712 is incorporated herein by reference. Loosely described here, a servlet is application code transferred from a first server to a second for execution on the second server.

Because servers are typically accessed orders of magnitude more frequently than a client workstation, maintaining the integrity of the server executing a servlet becomes even more critical than maintaining the integrity of the client executing an applet. Corruption on a server can spread quite rapidly to any number of clients. Should corruption pass among servers, the rate of corruption of clients can increase exponentially. The sandbox of the servlet is appropriately restricted.

Accordingly, a server receiving a servlet over a network relies on the security aspects instantiated in the Java® compiler, verifier and class loader as described generally below.

In the HotJava® browser mentioned above, the boundaries of the sandbox for an applet are as follows: An applet may not read, write or inquire into the status of any filesystem on the client workstation **150**. An applet from an server, say, **120***b* running on the workstation **150** can not access any other processor **120***a*, **120***n*, **150** over the network **160** other than its server processor **120***b*. An applet cannot load a library from either its server processor **120** or the workstation **150**. An applet cannot initiate the execution of a process. An applet cannot examine the properties of any resource on the workstation **150**.

To assist in the enforcement of the boundaries of the sandbox of an applet, the assignee of the instant invention has developed a suite of protocols for the development and

execution of applets: the Java® development environment (or Java® Development Kit). The development environment includes a number of packages ("lang," "io," "net," "util," "awt" and "applet"). To the extent an applet needs language, I/O, network, utility, windowing or application support, the application must resort to the methods available through the classes provided by one or some of the packages of the Java® development environment.

Of course, not all classes and methods can be anticipated. As such, the Java® development environment permits the construction of additional classes, methods and, ultimately, applets. However, the Java® development environment includes a compiler which performs a number of checks to ensure that the applet does not contain any security violations. For example, the Java® compiler does not permit pointers directly to memory. It strictly checks types. Access to an object must be through its public interface. The Java® compiler leaves memory management for the Java® interpreter, and the latter provides the former with no information on how it accomplishes the memory management.

A Java® verifier checks the code of an applet to ensure that the security of the Java® development system is intact. The verifier verifies the classfile, the type system checking, the bytecode and runtime type and access checking.

After the verifier passes the applet code, the Java® class loader checks the applet code to enforce namespaces based on the network source.

Within each method of a package, the code uses the security manager to provide a system resource access control mechanism. Any time an applet needs to access a particular resource, it uses a particular pre-defined method which provides checking as to whether the applet can access that particular resource.

In addition to the security aspects instantiated in the Java® compiler, verifier and class loader as described above, the receiving server invokes an improved security manager according to the invention. In addition to the security checks described above, the server's security manager identifies the network source of the servlet and implements a security policy based on the servlet's network source.

For example, the server's security manager may allow (or disallow) the execution of any servlet from a predetermined list of network sources. (In one embodiment, the authentication of the source of a servlet, particularly by digital signature, is the responsibility of the class loader.) Servlets from trusted servers identified by their digital signatures would be executed. Alternatively, the server's security manager may allow (or disallow) the execution of any unsigned servlet. As yet another alternative, the server's security manager may allow (or disallow) the execution of any signed servlet.

(Another embodiment of the invention, including Java® archives (JARs) is described in Table I.

TABLE I

```
How to sign and verify JAR files
Benefit: signature is detached from .class file; and if servlet has only one
.class file, then signed JAR file is just that .class file, plus the necessary
signature info
Scenario for signing JAR files:
    servlet == list of files
To sign a servlet, use standalone Java program to
    1. specify list of files
    2. specify method (CA-based or PGP-style)
    3. create JAR file, create signature, write signed JAR file
The sun.server.util.jar class library includes support for doing the signing
(as well as support for packaging up the files into a JAR file.)
    1. Jar j = createJarFile (list of files)
    2. h = j.hash (algorithm)
    3. sign = h. encrypt (algorithm)
    4. SignedJARFile = createJarFile (j, sig, cert, alg)
Verifying signed servlet, within server classloader
    1. If (SignedJarFile)
        1. extract JarFile, sign, cert, alg from SignedJarFile
        2. c = JarFile.hash (algorithm)     /* compute hash */
        3. pk = cert.getPublicKey ()
        4. h = sig.decrypt (algorithm)
        5. if (c == h) then /* signature is valid */
            is sig.ID on my list of trusted signatures?
            if yes, load servlet
A signed JAR file has these four fields.
The prototype uses PGP or X509 based certificate authorities.
SignedJarFile format
    * [ Jar file | .class file ]
    * signature
    * algorithm
    * [ X509 certificate | PGP public key]
```

More generally, the server's security manager may decide whether to execute a servlet based on some other characteristic of the applet.

For such servlets as the server allows to execute, the server must also decide what server resources the servlet may access. The Java® environment provides a default level of service. However, the server can decide to enlarge or even shrink the default set of resources to which the servlet has access.

TABLE II

	read (filename)		socket (host, port)
	strings immutable	access authorization	DNS name resolution
server 120a	Yes	Yes	Yes
server 120b	No	No	No
. . .			
server 120n	Yes	No	Yes

In one embodiment of the invention, the server maintains a configurable security policy on a group or per-server basis (as described above). The server maintains a list of configurable resources, a list of the configurable accesses possible for each resource and a cross-list of servers (or groups of servers) and the accesses they may have. Table II illustrates one such configurable security policy wherein server **120***a* is not trusted at all and all security checks are applied to any servlet having server **120***a* as its source, while server **120** is sufficiently trusted not to need access authorization for read()'s. Servlets from server **120***b* are completely trusted.

In the configurable policy embodiments, the namespace of the configurable embodiments supply the methods for the security manager's checks, typically by creating a subclass of the Java® environment's security manager.

In one embodiment, the security manager includes a means for notifying the system administrator of the server that a servlet desires a resource from which it is currently excluded. On notification, the system administrator may reconfigure the security policy to allow the servlet the desired access. (Whether some class of servlets may cause notification of the administrator can also be configurable.)

In another embodiment, signed servlets are completely trusted (as server **120***b* in Table II) and have full access to

the server. Unsigned servers, however, are blocked from executing HTTP requests and responses and inter-servlet communications. Unsigned servlets do not have access to the server's file system, properties files, dynamic configuration files, or memory management facilities.

FIG. 3 is a flowchart illustrating the flow of command according to an embodiment of the invention.

Of course, the client may likewise decide whether to execute application code (applet) loaded over the network based on the source or other characteristic of the applet.

Of course, the program text for such software as is herein disclosed can exist in its static form on a magnetic, optical or other disk, on magnetic tape or other medium requiring media movement for storage and/or retrieval, in ROM, in RAM, or in another data storage medium. That data storage medium may be integral to or insertable into a computer system.

What is claimed is:

1. A computer-implemented method for signing a first archive file, said method comprising:

selecting one of a plurality of security algorithms for use with said first archive file;

computing a signature for said first archive file according to said selected security algorithm; and

creating a new archive file comprising said first archive file, said selected security algorithm and said signature.

2. The method of claim 1, wherein said first archive file comprises a single file.

3. The method of claim 1, wherein said selecting comprises selecting a certificate-based algorithm for use with said first archive file.

4. A computer-readable medium containing instructions for performing a method for causing a computer system to sign a first archive file, said method comprising:

selecting one of a plurality of security algorithms for use with said first archive file;

computing a signature for said first archive file according to said selected security algorithm; and

creating a new archive file comprising said first archive file, said selected security algorithm and said signature.

5. A computer-implemented method for validating a signed archive file comprising an included archive file, an associated security algorithm and a signature, said method comprising:

extracting said included archive file, said associated security algorithm, and said signature from said signed archive file, wherein the associated security algorithm is selected from one of a plurality of security algorithms for use with said included archive file;

calculating a value for said included archive file according to said associated security algorithm;

extracting a corresponding value from said signature; and

determining said archive file as valid when said calculated value and said extracted corresponding value compare equal.

6. The method of claim 5, wherein said extracting of a corresponding value comprises conferring with a certificate authority to obtain a certificate for verifying said signature.

7. A computer-readable medium containing instructions for performing a method for causing a computer system to validate a signed archive file comprising an included archive file, an associated security algorithm and a signature, said method comprising:

extracting said included archive file, said associated security algorithm, and said signature from said signed archive file, wherein the associated security algorithm is selected from one of a plurality of security algorithms for use with said included archive file;

calculating a value for said included archive file according to said associated security algorithm;

extracting a corresponding value from said signature; and

determining said archive file as valid when said calculated value and said extracted corresponding value compare equal.

8. A computer system comprising:

a medium for data storage wherein is located a computer program for causing a computer system to validate a signed archive file comprising an included archive file, an associated security algorithm and a signature by

extracting said included archive file, said associated security algorithm, and said signature from said signed archive file, wherein the associated security algorithm is selected from one of a plurality of security algorithms for use with said included archive file;

calculating a value for said included archive file according to said associated security algorithm;

extracting a corresponding value from said signature; and

determining said archive file as valid when said calculated value and said extracted corresponding value compare equal; and

a CPU couple to said medium, for executing computer programs.

9. A medium for data storage comprising:

a first program for creating a first archive file;

a second program for

selecting one of a plurality of security algorithms for use with said first archive file;

computing a signature for said first archive file according to said selected security algorithm; and

creating a new archive file comprising said first archive file, said selected security algorithm and said signature; and

a CPU, coupled to said data storage medium, for executing computer programs.

10. A computer-implemented method for signing a first archive file, said method comprising:

selecting one of a plurality of security algorithms for use with said first archive file;

computing a signature for said first archive file according to said selected security algorithm; and

creating a new archive file comprising said first archive file, said selected security algorithm, said signature, and a certificate.

* * * * *